Minimum Ratio Contours on Surface Meshes

Andrew Clements Hao Zhang
GrUVi Lab, School of Computing Sciences
Simon Fraser University, Burnaby, BC, Canada
aclement,haoz@cs.sfu.ca

Abstract

We present a novel approach for discretely optimizing contours on the surface of a triangle mesh. This is achieved through the use of a minimum ratio cycle (MRC) algorithm, where we compute a contour having the minimal ratio between a novel contour energy term and the length of the contour. Given an initial contour, we seek to find the optimal contour within a prescribed search domain. The domain of admissible contours is modeled by a weighted acyclic edge graph, where nodes in the graph correspond to directed edges in the mesh. The acyclicity of this graph allows for an efficient computation of the MRC. To further improve the result, the algorithm may be run on a refined mesh to allow for smoother contours that can cut across mesh faces. We demonstrate the effectiveness of our algorithm in postprocessing for mesh segmentation.

1. Introduction

Mesh feature extraction and segmentation are important prerequisites to several problems in geometry processing, e.g., mesh parameterization, morphing, and 3D shape matching and recognition. Active contours [14], also known as snakes, have been successfully applied to feature extraction and segmentation for image analysis. Recently, they have been extended to work on surface meshes [3, 16, 13] and applied to mesh feature analysis and segmentation, e.g., in postprocessing for mesh scissoring [10, 17].

The snake method uses gradient descent to reach a local minimum of a boundary cost function, which is typically given by a combination of the external and internal energy defined for a contour. The cost function represents a *total energy*, which, if optimized globally, yields a trivial solution. Thus the fact that snakes can often adapt to image or geometric features well is more of an artifact of the local nature of the optimization procedure and not necessarily the result of a snake having reached a state that is close to the global minimum [22].

In this paper, we advocate the use of an *energy ratio* as the cost function, in a global optimization setting. Although the energies making up the ratio can vary, we focus on the mesh segmentation problem and define the numerator weight to combine the internal (for smoothness) and external (for feature adaptation) energies of a contour and the denominator weight as contour length. In addition to having a different energy to minimize, our algorithm differs from previous implementations of mesh snakes [3, 4, 5, 16, 18, 13] in that a discrete, graph-based search is used to find a globally optimal contour. Our approach operates directly on the surface mesh and not in a parameter domain [16]. It is efficient and is flexible and general enough to allow the generation of both closed and open curves, curves cutting across mesh faces, and the incorporation of both hard and soft constraints. Topological changes can also be detected and handled quite easily.

The rest of the paper is organized as follows. In Section 2, we briefly survey previous works on optimizing contours on images and surface meshes. Section 3 gives an overview of our approach. In Section 4, we define the minimum ratio cycle and path problems and cover the relevant theory. In Section 5, we describe our minimum ratio contour algorithm in details, focusing on the construction of an acyclic graph as our search space and the definition of our new energy measure appropriate for our target application. In Section 6, we describe a refinement scheme which would allow the contour to cut across mesh faces; this improves the smoothness quality of the final contour. Experimental results are provided in Section 7 and in Section 8, we conclude and discuss possible future investigations.

2. Previous work

One of the most well-known boundary-based image feature extraction and segmentation methods uses the active contour model, first introduced by Kass et al. [14]. The movement of a snake over time is controlled by minimizing its associated energy, which is a combination of an internal energy and an external energy term. Minimizing the internal

energy tends to shorten the snake while keeping it smooth. External energy serves to attract the snake toward features, whose definition is application-dependent.

A snake can be represented either explicitly or implicitly. Explicit models store exact locations, called snaxels, the snake passes through, e.g., in [1, 2, 3, 13]. Amini et al. [1] use dynamic programming to optimize image snakes in the presence of hard constraints. In this case, every snaxel can move to at most m different positions and the optimal snake is found in $O(nm^3)$ time at each step, where n is the number of snaxels. Williams and Shah [23] in turn propose a greedy algorithm, which may not find the optimum, but runs in $O(nm)$ time. Implicit methods, e.g., [4, 5], consider the snake positions to lie in the zero level set of a function defined over an ambient space. They need not track topological changes explicitly, but do not allow intuitive user control of the snake, in contrast to explicit models.

An early work on mesh snaking is due to Milroy et al. [18], which uses a greedy algorithm to segment a wrap-around model. Jung et al. [13] adapts the fast greedy approach of Williams and Shah [23] in a straightforward manner to triangle meshes. Lee and Lee [16] take a different approach, by first locally parameterizing the snake from the surface of a 3D mesh onto a 2D domain, then evolving the snake in the plane using techniques developed for images, and finally mapping the 2D snake back onto the 3D mesh. The parameterization and image snake movements are carried out patch by patch, which overlap each other to ensure smoothness of the overall mesh snake across patch boundaries. To avoid difficulties due to parameterization artifacts, Bischoff and Kobbelt [2] implement an explicit, parameterization-free mesh snaking algorithm, where snaxels may lie on the lines of a uniform grid, so that topology control and self-intersections of the snake are handled in an efficient and controlled manner. This technique has later been extended to triangle meshes [3]. However, as with all active contour models, the final snake is only locally optimal, so many initial snakes, close to the desired contour, need to be inserted manually on the mesh.

Cox et al. [9] first model the image segmentation problem as a *global* minimum ratio problem, where a directed cycle over the image grid that minimizes a ratio of the cost of the perimeter of the segmented region to the benefit assigned to its enclosed interior is sought. This is similar in spirit to the well-know normalized-cut approach to image segmentation [20]. Of more interest is a similar approach due to Jermyn and Ishikawa [12], which finds an optimal region by minimizing the ratio of an energy flow across the boundary of the region over the internal boundary energy, e.g., boundary length. Green's Theorem can be used to show that information about the interior region, e.g., its homogeneity, can also be incorporated into the minimization. Wang et al. [22, 21] solve a minimum *mean* cycle (MMC) problem on an undirected graph derived from an image and find a globally optimal contour which passes through a precomputed set of edge fragments. Note that the MMC problem differs from the minimum *ratio* cycle problem solved by Jermyn [12] in that the denominator is the number of edges in the cycle and not its total length.

A scissoring technique that resembles our optimization based approach is given by Funkenhouser [10]. As with our approach, they apply a graph based optimization in which an optimal contour is found within a given search region. The search region is usually a small neighbourhood surrounding an initial segment of a contour given by a user, and the optimal contour is found using Dijkstra's algorithm. Fundamentally, their method is similar to an active contour approach, since the cost function that they are minimizing is a *total energy*. This differs from our approach, since we advocate the use of a *ratio energy*, which removes the bias towards shorter contours.

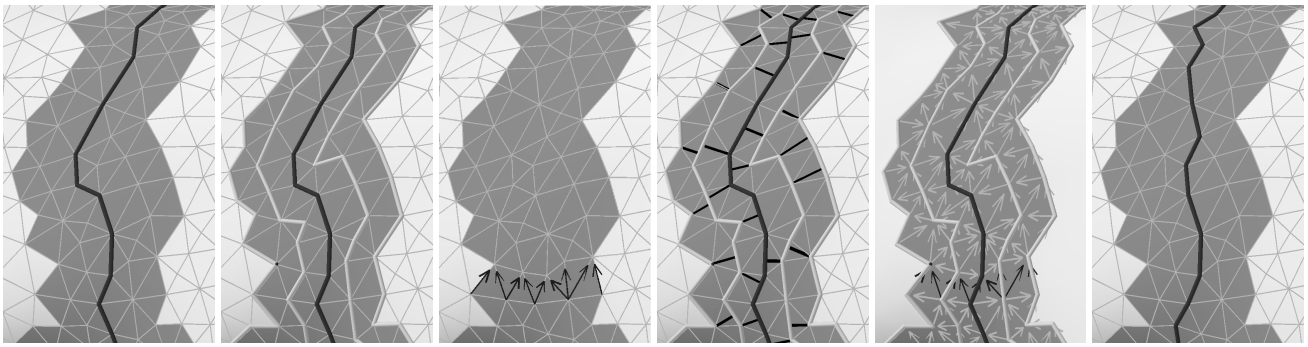
3. Overview of our approach

One of main challenges in applying the minimum ratio idea to contour optimization on an irregular-grid surface mesh is to come up with an appropriate search space, which will be a weighted graph. We wish to ensure that the globally optimal cycle is simple (no self-intersections), efficient to compute, represents a smooth and perceptually meaningful segmentation, and that it wraps around a 3D model if desired. We accomplish all these with a novel construction of an acyclic graph based on a novel energy definition. Our approach has the advantage that it decouples contour optimization from feature extraction and selection, e.g., compared to [17, 21]. However, if so desired, both hard and soft constraints may be enforced within the same framework.

In Figure 1, we illustrate the major steps of our algorithm for optimizing a contour that only passes through mesh edges. We describe these steps below.

1. **Search space — Figure 1(a) and (b):** Given an initial contour on a mesh, a search space consisting of a band, with a user-specified width, surrounding the initial contour is first constructed, as shown in Figure 1(a). The search band is computed via face dilation, starting from the initial contour, as shown in Figure 1(b). Also obtained are the "strip boundaries", shown in light gray, which are separated by single strips of triangles and used to refine the search space when constructing the search graph.

2. **Edge cut — Figure 1(c):** Next, a minimal edge cut of the search band is found, as shown in Figure 1(c). Each mesh edge along the cut is duplicated, resulting in pairs of corresponding source and destination edges for our subsequent search. This is to ensure that the resulting



(a) (b) (c) (d) (e) (f)

Figure 1. Major steps of our algorithm. (a) Given an initial contour (dark gray), a search band around it with a user-defined width d (here $d = 2$) is found. (b) The 2-ring search region is found by dilation from the initial contour, resulting in a set of strip boundaries, drawn in light gray. (c) An edge cut through the search space is obtained. (d) Gate segments are inserted between adjacent strip boundaries; they aid in the determination of which directed mesh edges to include in the acyclic graph. (e) The acyclic edge graph is constructed. Each edge in the edge cut is duplicated and serves as source and target nodes in the acyclic graph. (f) Optimal ratio contour found by running the MRP algorithm once for each directed edge in the edge cut.

contour wraps around the band as the initial contour does; in general, an minimum ratio contour (MRC) over the search space may not possess that property.

3. **Acyclic search graph — Figure 1(d) and (e):** Within the search space, we build a weighted acyclic graph (source to destination) whose *nodes* (vertices) are directed mesh edges, as shown in Figure 1(e). Each "edge" in the acyclic graph, which we refer to as an *arc* to distinguish it from mesh edges, has a denominator and a numerator weight. The former models contour length while the latter ensures contour smoothness and feature adaptation; these are described in Section 5.5.

   An intermediate step in the construction of the acyclic graph is the computation of *gate* segments which connect adjacent strip boundaries, as shown in 1(d). Intuitively, the gates are located at constrictions between adjacent strip boundaries and they help define a linear ordering of mesh edges along the strip of triangles sandwiched between adjacent strip boundaries.

4. **Optimization — Figure 1(f):** To compute a *minimum ratio path* (MRP), shown in Figure 1(f), between a pair of corresponding source and destination nodes s and t in the search graph, the Bellman-Ford algorithm [8] is run a number of times to *linearly search* for an appropriate edge weight offset which results in a zero-length shortest path between s and t.

The acyclicity of the search graph allows us to carry out the Bellman-Ford algorithm in linear time. The overall complexity of our algorithm is $O(mnd)$, where n is the number of mesh edges within the search space, d is the width of the search band, and m is an upper bound on the number of linear search steps performed for each (s, t) pair. As confirmed by both our experiments and related work on image contour extraction [12], m is typically a small constant (no more than six in all of our experiments).

Both the graph connectivity and arc weights rely on an analysis of mesh geometry; this step is application-dependent. For our target application, mesh segmentation, we follow the *minima rule* from cognitive studies [11], which necessitates the computation of principal curvatures and directions over a mesh. We adopt the tensor estimation of Cohen-Steiner and Morvan [7]. As suggested, to account for noise present in the input mesh, several iterations of Gaussian smoothing are applied to the curvature tensors.

Restricting a contour to be along mesh edges has its pros and cons. On one hand, the smoothness of the contour is limited by the mesh resolution and may be compromised by possible connectivity artifacts. On the other hand, cutting through mesh faces may add sliver triangles to the segmented mesh and also increase computation time. If desired, once the contour has settled into position, we can refine the mesh, as explained in Section 6, and run our search again to produce an optimal contour with increased resolution and smoothness. Additionally, our approach allows

- **Open curves:** As our algorithm essentially reduces an MRC problem to a number of MRP problems, open curves are straightforward to handle.

- **Incorporation of hard constraints:** It is easy to force an optimal solution to pass through a set of constraint edges, e.g., salient feature edges over the mesh. As an acyclic graph induces a partial order on its nodes, an MRP problem from source s to target t can be decomposed into a sequence of MRP problems for which each constraint edge becomes both the source and target nodes of two adjacent MRP problems.
- **Incorporation of soft constraints:** Proximity to soft constraint edges can be included in the definition of numerator energy at the arcs in our acyclic graph.

4. The minimum ratio path problem

Given a directed search graph $G = (V, E)$, associate two functions $f, g : E \rightarrow \mathbf{R}$ to the arcs (edges) of G. Consider a sequence π of consecutive arcs in G, which may or may not form a cycle, define the *numerator weight* of π to be

$$\chi(\pi) = \sum_{e \in C} f(e)$$

and the *denominator weight* to be

$$l(\pi) = \sum_{e \in C} g(e).$$

Let $\mathcal{C}$ be the set of all directed cycles in G. The *minimum ratio cycle* (MRC) problem is one that seeks a cycle $\pi^* \in \mathcal{C}$ which minimizes the ratio $\tau(\pi) = \chi(\pi)/l(\pi)$. That is,

$$\pi^* = \text{argmin}_{\pi \in \mathcal{C}} \tau(\pi) = \text{argmin}_{\pi \in \mathcal{C}} \frac{\chi(\pi)}{l(\pi)}.$$

The related *minimum mean cycle* problem [22] replaces the denominator $l(\pi)$ by the arc count $\#(\pi)$. Both problems can also be defined for the set of paths from a source vertex s to a destination vertex t; we refer to these problems as the MRP and MMP problems, respectively.

To find the MRP in G, let us observe: the ratio of any path π from s to t in G is reduced by $\lambda \in R$ when a new set of numerator weights $f'(e)$, given by

$$f'(e) = f(e) - \lambda \cdot g(e), \forall e \in E(G)$$

are used to replace f. This is easy to see, as

$$\frac{\sum_{e \in \pi} f'(e)}{\sum_{e \in \pi} g(e)} = \frac{\sum_{e \in \pi} \Big[f(e) - \lambda \cdot g(e) \Big]}{\sum_{e \in \pi} g(e)} = \frac{\sum_{e \in \pi} f(e)}{\sum_{e \in \pi} g(e)} - \lambda$$

Since the ratio of each path in G is lowered by λ, the MRP using weights f' remains unchanged from the MRP in G using the original weights f.

To solve the MRP problem in G, we can then search for the smallest λ^* such that the MRP in G, with the same source s and destination t, weighted by $f^* = f - \lambda^* \cdot g$, has a ratio value of zero. This is in turn equivalent to determining whether G weighted by f^* has a zero-weight path from s to t and no negative-weight paths from s to t. Determining whether a negative weight path from s to t exists can be accomplished by running the Bellman-Ford algorithm on G. Thus to find λ^*, either a binary or linear search over λ can be performed. Although the time complexity of binary search is lower than that of linear search (a pseudo-polynomial time bound can be obtained for linear search [12] if the denominator and numerator weights are integral), in practice linear search is found to more efficient [12] and this is what we use for our MRP problem.

The Bellman-Ford algorithm is needed above to test whether G has a negative-weight path from s to t for a given λ. In general, this would take $O(|V| \cdot |E|)$ time [8]. However, for our purpose, the search graph G will be acyclic and the complexity of the procedure is then reduced to linear-time, more specifically, $O(|V| + |E|)$.

Wang and Siskind [22] consider the undirected version of the MMC and MMP problems and solve them by first transforming G into an augmented graph G' and then reducing the problem of finding a negative cycle in G to that of finding a minimum weight perfect matching (MWPM) in G'. We have not considered this approach since in general the cost of performing MWPM is $O(|V| \cdot |E| \log |V|)$, which is much higher than the linear-time algorithm we have been able to obtain for our specialized graph model.

5. Minimum ratio contour algorithm

In this section, details of each stage of our minimum ratio contour algorithm are given. Refer to Figure 1 for an illustration of some of these steps.

5.1 Search space

Given an initial contour C_0, we construct a search space, which is a band to approximate a d-ring neighborhood about C_0. We successively perform face dilation starting from C_0. If the active boundary of the dilation would intersect itself, we disallow addition of that face, in order to ensure that the search space is homotopic to a solid torus.

5.2 Finding an edge cut

A contour which traverses around the search band must pass through an edge in an edge cut of the band. Instead of implementing a full-fledged graph min-cut algorithm, we perform a breadth-first-search (BFS) on mesh faces within the search space due to the special banded shape of the

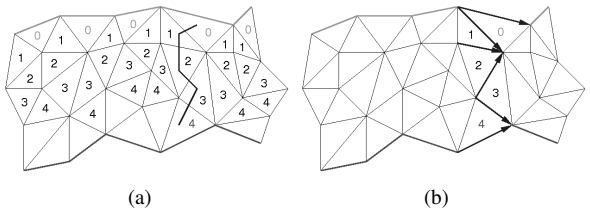


Figure 2. Finding an edge cut via BFS. (a) A minimal sequence of faces is found. (b) Corresponding edge cut.

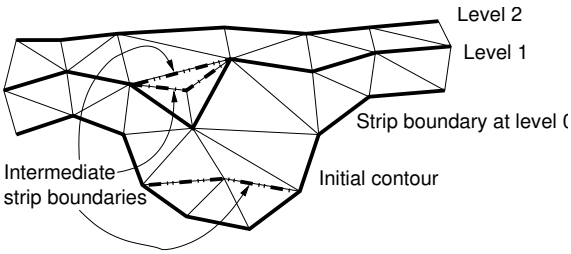


Figure 3. An initial contour and the strip boundaries, on one side, obtained.

search space. Specifically, we label each face having an edge on the inner boundary of the search band 0 and then use BFS with face adjacency to label the remaining faces in the search band, stopping when a face on the outer boundary is encountered. Now a minimal sequence of connected faces from the inner to the outer boundary is obtained by backtracking. The corresponding edge cut consists of edges which are adjacent to two faces in the minimal face sequence, plus one boundary edge, as shown in Figure 2.

5.3 Constructing the acyclic graph

To construct an acyclic search graph G in the search region, we wish to establish a *flow direction*, which mimics the general direction of the initial contour. We then orient each undirected mesh edge within the search space; the orientation chosen is the one which is aligned with the flow direction. Care must be taken to ensure that vortexes (circular flow patterns) do not occur within the search space.

Strip boundaries: To facilitate construction of the vortex-free flow, we refine the search space into a series of *strip boundaries*. Each strip boundary is a progression of mesh edges, starting at an edge in the edge cut and ending at an edge in the edge cut. Specifically, the initial contour C_0, broken up at the edge cut, induces one of the strip boundaries. The set of all strip boundaries provide a complete cover of the mesh vertices within the search space. No two strip boundaries cross each other although they may overlap. We shall describe their construction later.

Distance function along strip boundaries: We define a *monotonic distance function* $\mathcal{F}$, in the range [0,1], along each strip boundary. It measures some form of a distance between a given point along a strip boundary to the starting point of the strip boundary. The flow direction is dictated by the distance functions.

Edge orientation and graph construction: By our definition of strip boundaries, every mesh edge within the search space has its two vertices either on the same strip boundary or on two adjacent strip boundaries. To orient each undirected mesh edge (a, b), we consider the distance or $\mathcal{F}$ values at its endpoints. If $\mathcal{F}(a) < \mathcal{F}(b)$, the edge is oriented from a to b and added to the graph G as a *node*. If $\mathcal{F}(b) < \mathcal{F}(a)$, the edge is oriented from b to a and added to G. Otherwise, the edge lies along an "iso-level", and it is not added to G. An arc, connecting two adjacent nodes (a, b) and (b, c) in G, is added to G if and only if the angle between the directed mesh edges (a, b) and (b, c) does not exceed a threshold of $\pi/2$. By construction, the resulting directed graph G must be acyclic. Also, each strip boundary induces a directed path in the search graph G.

Construction of strip boundaries: We construct strip boundaries by levels. At level 0, we have the strip boundary corresponding to the initial contour. The i-th level strip boundaries (there are two of them, one to each side of the initial contour) are composed of the i-ring neighborhood vertices of the initial contour. If any such strip boundary self-intersects, we remove the smaller intersection piece to ensure a non-intersecting strip boundary. As a result, not all vertices in the search space may lie on a strip boundary and it may be necessary to add additional, refined boundaries. This is achieved by successively inserting intermediate strip boundaries until all vertices are covered by the strip boundaries. In Figure 3, we show the set of strip boundaries obtained in an artificial yet non-trivial configuration.

Distance function based on arc lengths: The distance function used could be based on arc lengths, where the $\mathcal{F}$ value at a point p along a strip boundary C would be the distance from the starting point of C to p, divided by the total length of C. This simple distance function may encounter problems if the length of adjacent strip boundaries differ greatly, which can result in flow directions which do not mimic the general flow of the initial contour.

Distance function based on gates: A better approach is to enforce a set of constraints between the distance functions of adjacent strip boundaries. A *gate* is a line segment which connects two adjacent strip boundaries. The distance function at the endpoints of a gate segment (p, q), where p is on strip boundary C_1 and q is on strip boundary C_2, are

required to be equal; that is $\mathcal{F}_{C_1}(p) = \mathcal{F}_{C_2}(q)$. As a gate between adjacent strip boundaries C_1 and C_2, we have chosen pairs of points (p, q), $p \in C_1$ and $q \in C_2$, such that the closest point on C_2 to p is q, and the closest point on C_1 to q is p; see Figure 1(d). The motivation for choosing such gate segments is that such pairs of points (p, q) correspond to narrows between the strip boundaries. The narrows are natural places to insert gates. In practice it has been observed that the number and locations of the gates are sufficient in establishing adequate flow directions.

5.4 Optimization

Finally, nodes corresponding to edges in the edge cut are duplicated. Let us refer to one set of duplicated nodes as *source*, and the other as *dest*, for destination nodes. We only allow outgoing arcs from nodes in *source* and incoming arcs to nodes in *dest*. Now a progression of nodes in G starting at a node in *source* and ending at its duplicate node in *dest* corresponds to a contour (or cycle) on the mesh.

After constructing the acyclic graph, whose weights will be defined in the next section, we run the MRP algorithm as mentioned in Section 4 once for each pair of duplicated nodes, one from *source* and the other from *dest*. We take the MRC as the one obtained as the minimum of the set of MRPs, with the source and destination nodes merged.

5.5 Energy definition

Given an arc $r = (e_1, e_2)$ in the constructed acyclic graph, corresponding to an ordered pair of directed mesh edges $e_1 = (u, v)$ and $e_2 = (v, w)$, the denominator weight of r models length and is simply defined as

$$g(r) = \frac{1}{2}(|e_1| + |e_2|),$$

where $|\cdot|$ is the Euclidean norm. The numerator weight should serve to penalize bending between e_1 and e_2 and to attract them to the desired features, which we discuss below.

- **Bending:** In contrast to smoothness energies previously defined for mesh snakes [16, 13], which model second-order derivatives in the 3D ambient space, we wish to consider only bending "inside" the surface. This notion corresponds to the idea of "straightness" of Polthier and Schmies [19]. Instead of using left and right curve angles to model straightness, we project e_1 and e_2 onto the tangent plane at v and measure smoothness in the tangent plane; this allows us to combine bending and contour steering into a single term.

- **Feature adaptation:** For segmentation using the minima rule [11], which states that segmentation boundaries should consist of surface points at *negative minima of principal curvatures*, the feature energy at a vertex is related to its minimum principal curvature κ_{min}. More specifically, the negative curvature minima are used to attract the contour[1] and no preference is given to points with $\kappa_{min} \geq 0$.

- **Contour steering:** There is no provision in the minimum ratio objective to produce short contours. This is seen as an advantage of the approach as the optimal contour can adapt better to features [22]. However, with the traditional snake energy, e.g., [16], in the numerator, the MRC has a tendency to wind through the search region to attract itself to features. As long as the straight portion of the contour still dominates the turning portion, the ratio will be kept small.

  We have found a remedy for this by steering each edge of the contour, bringing it to alignment with a consistent flow direction. In the context of mesh segmentation, we have chosen that direction to be the direction of maximal principal curvatures.

Combining the above considerations, we define the numerator weight of the arc $r = (e_1, e_2)$ to be

$$f(r) = \begin{cases} \frac{1}{2}\kappa_{min}(v) \cdot \Big[|e_1| \cdot \cos\theta_1 + |e_2| \cdot \cos\theta_2\Big], \\ \qquad\qquad\qquad\qquad \text{if } \kappa_{min}(v) < 0; \\ \frac{1}{2}\kappa_0 \cdot \Big[|e_1| \cdot (1 - \cos\theta_1) + |e_2| \cdot (1 - \cos\theta_2)\Big], \\ \qquad\qquad\qquad\qquad \text{otherwise,} \end{cases} \tag{1}$$

where θ_1 (resp. θ_2) is the angle between the projection of the vector e_1 (resp. e_2) in the tangent plane and the maximum principal curvature direction $\vec{p}_{max}(v)$, as shown in Figure 4. Note that the tangent plane at v is spanned by $\vec{p}_{max}(v)$ and $\vec{p}_{min}(v)$. As we do not consider an elliptical region, where $\kappa_{min} > 0$, to be a feature region for segmentation, we replace κ_{min} by $\kappa_0 > 0$, which is a large constant, in that case.

As we can see, bending is automatically incorporated since minimizing $(|e_1| \cdot \cos\theta_1 + |e_2| \cdot \cos\theta_2)$ has the effect of reducing the angle between e_1 and e_2, in the tangent plane. Also, it is possible to introduce free parameters as exponents in (1) for a trade-off between the two factors.

Finally, for a contour or path π, formed by a progression of arcs, its energy ratio is $\tau(\pi) = \frac{\chi(\pi)}{l(\pi)}$, where

$$l(\pi) = \sum_{r \in \pi} g(r) \quad \text{and} \quad \chi(\pi) = \sum_{r \in \pi} f(r).$$

[1] Obviously, when ridge lines are to be detected, positive curvature maxima would become relevant.

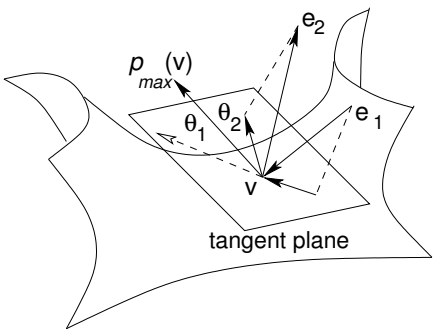
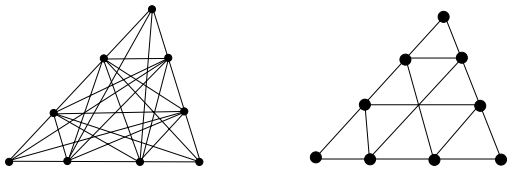


Figure 4. Numerator weight is computed by projecting edges onto the tangent plane.



(a) Refinement: $k = 2$. (b) Subdivision.

Figure 5. (a) Each line segment shown is an undirected chord. (b) Subdivision with the same number of Steiner points does not result in sufficient smoothness.

6. Optimization of refined contours

In order to overcome the limitation that our contour may only traverse along the edges of a mesh, we use a refinement scheme similar to that of Lanthier et al. [15]. Our refinement permits segments of the contour to cut across faces so as to avoid connectivity artifacts, improving the resolution and smoothness quality of the resulting contours.

6.1 Refinement scheme

Unlike mesh snakes, e.g., [16, 3], which model continuous movements of a contour, we discretize the process. Specifically, we insert k equally spaced Steiner points along each edge of the original mesh. For each triangle T, we define a *directed chord*, a term chosen to distinguish it from the original directed edges of the mesh, to be an ordered pair (p, q) for which one of following is true:

- p and q are adjacent points, either a vertex of T or a Steiner point, along the same edge;
- p is a vertex of T and q is a Steiner point on the triangle edge opposite to p;
- p is a Steiner point and q is another Steiner point that lies on a different edge of T.

Each refined contour is composed of a directed sequence of chords. The larger the value of k, the smoother a refined contour can be. But we have found that a small k, e.g., $k = 2$, is quite sufficient to produce high-quality contours. Figure 5(a) shows the set of chords (undirected) in a triangle, with $k = 2$. In contrast, refinement using subdivision, as shown in Figure 5(b), may not result in sufficiently smooth contours. Note that our setting is a bit different from that of Lanthier et al. [15], since we work on a chord graph.

It is not hard to verify that there are $6k^2 + 12k + 6$ directed chords per triangle. If we join two chords as long as they are connected, then the size of the search graph may be quite large, even for small k. Although the acyclic graph construction can reduce that complexity dramatically, we adopt two simple strategies first to prune the refined search graph. Specifically, we remove any arc that

1. connects two chords interior to the same triangle, or
2. connects two chords that sustain an angle exceeding a certain threshold, i.e., when the bending is too great.

6.2 The acyclic graph and optimization

Our optimization procedure for the refined case follows the same general structure as the regular case, described in Section 5, with only slight modifications. After identifying the search band, we consider the refined chord graph (after pruning) to find an edge cut.

Let γ be a path, in the original mesh graph, originating at a vertex situated on the inner boundary, and terminating at a vertex on the outer boundary of the search band. We take as an edge cut in the refined chord graph all chords which originate on one side of γ and terminate at some point, a mesh vertex or a Steiner point, along γ. In practice, we take γ to be a shortest (in graph distance) path from the inner to outer boundary of the search band. Note that although the number of edges in the resulting cut may not be minimal, it is nevertheless a close approximation.

The construction of the acyclic graph in the refined case proceeds in the same manner as in the regular case, except that directed chords in the search space need to be oriented instead of mesh edges. To find the principal curvatures and directions, as well as distance values for Steiner points, we linearly interpolate along a mesh edge those values that have already been estimated at the mesh vertices.

7. Experimental results

Results obtained from our MRC algorithm are shown in Figures 6, 7, 8, 9, and 10. All experiments were performed on a commodity desktop computer. Statistics about the search space and the constructed acyclic graphs can be found in Table 1. The timing statistics for our algorithm can be found in Table 2. Of special interest in Table 2 is the recorded maximum number of linear search steps required for each MRP problem (one per pair of source and destination nodes). In all of our experiments, this number is at most 6, attesting to the efficiency of the linear search. For each experiment, the user is required to select a search space width. The significance and effects of differing widths are explained throughout this section.

As seen in Figures 6(b) and 7(b), connectivity artifacts are common in the regular, unrefined case. We find that refining the mesh, with $k = 1$ or 2, can render the resulting contours sufficiently smooth, but at an increased computation cost. The proper choice of k to ensure smoothness depends on the resolution of the given mesh and screen resolution. Typically, the lower the mesh resolution, the larger k needs to be. We run our algorithm on the same initial contour under three different optimization conditions: regular, refined with $k = 1$, and refined with $k = 2$, on the horse model and show results in Figure 6. As we can see, the algorithm converges to essentially the same final contour, but with different degrees of smoothness; this is only restricted by the given search graph. Thus our MRC algorithm is seen to be stable against the optimization conditions.

For the results shown in this paper, we typically first run the regular, unrefined optimization with an initial contour and possibly with a large search band width. Once the contour has settled, a refined optimization is applied, with a smaller neighborhood width, to obtain a smooth final contour. This is more efficient than conducting the entire search using the refined optimization.

Figures 7,8, and 9 give good examples to show possible differences between our MRC algorithm and mesh snakes. Consider first the case of the ring finger, shown in Figure 7. Note that the initial contour is far away from the desired feature, the final contour shown in Figure 7(b). Geometrically, the finger is smooth with its diameter continuously narrowing down to the finger tip. Thus a mesh snake would converge to a trivial solution, unless it is specifically instructed to inflate [6]. This is not the case however when the energy to minimize is a ratio. The ability of our MRC algorithm to adapt to mesh features allows it to locate the desired feature, provided that part of the desired feature lies in the initial search space. After two steps of MRC search, the optimal contour converges to the correct cut.

Figure 8 highlights the effect of the search space size on the resulting contour. With a search band of width 6, the resulting contour converges to that shown in Figure 8(b). This contour is the minimum ratio contour within the given search region. However, if the search band width is increased to 12, a lower ratio contour occurs in the search region, and the resulting contour jumps to the one shown in Figure 8(c). This is unlike a mesh snake, which would not be able to cross the large concave region present at the base of the thumb.

Again, Figure 9 again demonstrates how our MRC algorithm can avoid or bypass local minimums that a mesh snake would not. The initial contour lies around the torso of the headless model, part of which lies amidst a series of grooves present in the abdomen. By using a large neighborhood width of 12, the MRC algorithm is able is able to avoid the locally optimal contours passing through the grooves in the abdomen, and locate the much more significant final contour, shown in Figure 9(b).

It should be noted that if part of the desired contour does not lie within the search space, the resulting optimized contour will not be the desired one. Iterating the MRC algorithm will not help, unless an intermediate contour is found which has a lower energy than the inital contour, but a greater energy than the desired contour. To illustrate where repeated iterations are useful, consider the venus model in Figure 10(a), where there are a pair of contours whose initial positions are far from their desired final positions (segmenting the eyes). Note that part of the initial contours lie near low energy, concave regions of the eyes. These regions are included in the search space if using a small width of 2, and serve as an anchor when iterating the MRC algorithm. The result after one iteration is shown in Figure 10(b), and it takes five iterations to converge to the desired contour, shown in Figure 10(c).

8. Discussion and future work

In this paper, we present an algorithm for minimizing an energy ratio globally over a prescribed search band surrounding a given initial contour. The energy to be minimized is a ratio of a combination of straightness and feature weight of a contour to contour length. It is thus scale-independent, feature-conforming, and does not have any bias toward either short or long contours, nor any tendency to produce a large enclosed region, as with normalized cuts [20]. Our optimization procedure is efficient and produces promising results in postprocessing for mesh segmentation. This is made possible with a novel construction of an acyclic search graph and its associated edge weights.

While for a locally-optimizing mesh snake to locate a desired contour C, many initial snakes close to C may need to be specified manually, our approach requires only one initialization, provided that the search space encompass the desired contour. In this sense, the minimum ratio cycle ap-

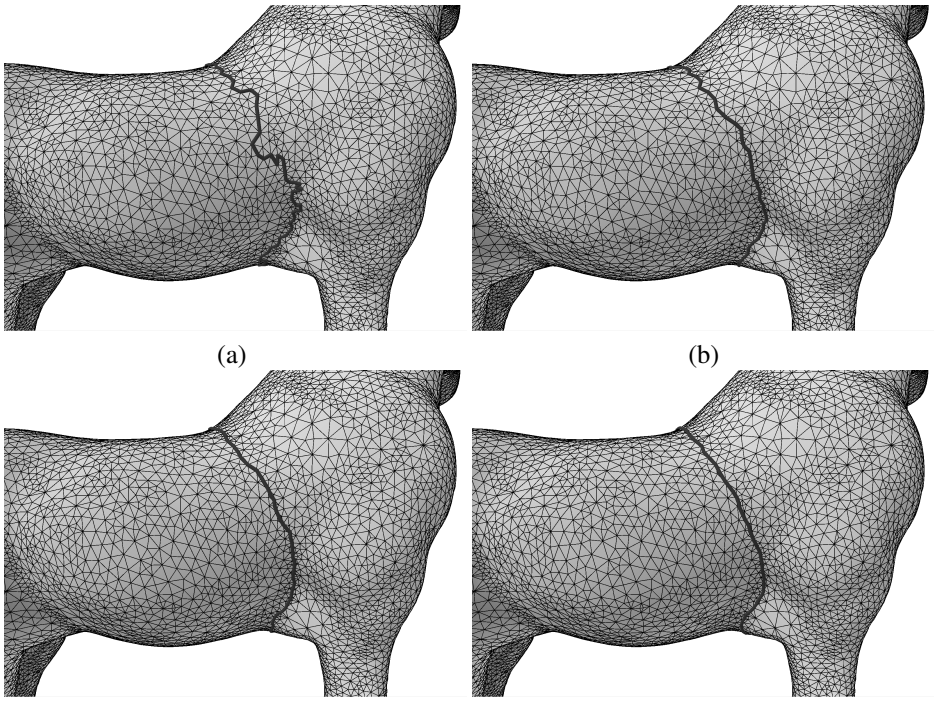


Figure 6. (a) An initial contour. (b) Optimal contour found using regular optimization; the connectivity artifacts are visible. (c) After optimization on refined graph, with $k = 1$. (d) After optimization on refined graph, with $k = 2$. Evidently, higher level of refinement results in smoother contours. These results also demonstrate the stability of our MRC algorithm.

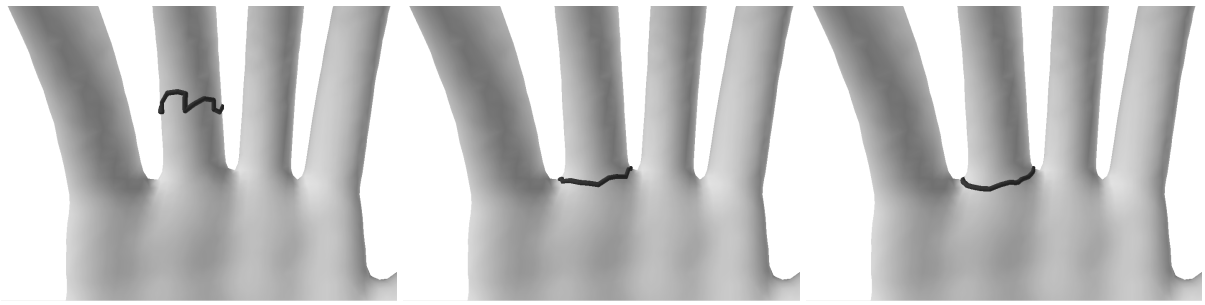


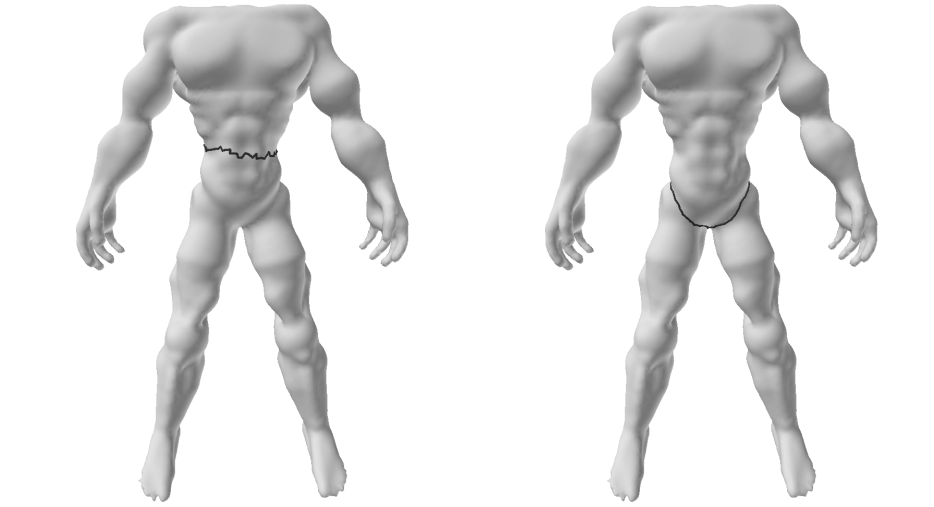
Figure 7. (a) Initial contour. A large search band of width 6 is used in order to make the contour jump to an optimal position. (b) Result after regular optimization; there is some roughness in the final contour as a connectivity artifact. (c) Result after refined optimization has increased smoothness.

proach is more automatic and requires less user intervention. As for efficiency, we expect our algorithm, especially in the refined case, to be more expensive, by some constant, than an individual mesh snaking session. But one needs to keep in mind that many such sessions may be required to obtained the desired result.

For future work, we would like to investigate whether it is possible to introduce a bias to-wards short contours in the minimum ratio framework, to add more flexibility to this approach. Furthermore, we would like to extend the search space to over the whole mesh surface. To be able to achieve reasonable speed for such a global optimization, a multiresolution approach may become necessary.

(a) (b) (c)

Figure 8. (a) Initial contour. (b) Using a neighborhood size of width 6 results in a locally optimal contour. (c) By increasing the size of the band to 12, to contour jumps to a better minima. After the contour has settled using regular optimization, refined optimization with $k = 2$ is used to produce the results in (b) and (c).



(a) (b)

Figure 9. (a) Initial contour. (b) Several iterations of with a large neighborhood size of width 12 allows the initial contour to realize a better position. Note that the contour has jumped across many local minima to find a better minima.

References

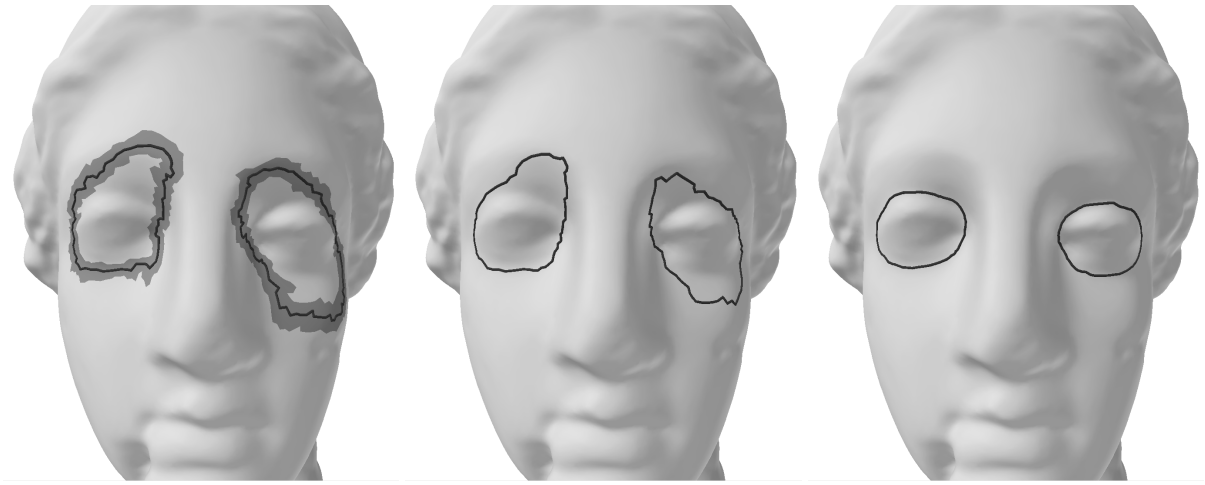
[1] A. Amini, S. Tehrani, and T. Weymouth. Using dynamic programming for minimizing the energy of active contours in the presence of hard constraints. In *IEEE Int. Conf. on Computer Vision*, pages 95–99, 1998.

[2] S. Bischoff and L. Kobbelt. Parameterization-free active contour models. *The Visual Computer*, 20:217–228, 2004.

[3] S. Bischoff, T. Weyand, and L. Kobbelt. Snakes on triangle meshes. *Bildverarbeitung für die Medizin*, pages 208–212, 2005.

[4] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *Int. Journal on Computer Vision*, 22(1):61–79, 1997.

[5] L.T. Cheng, P. Burchard, B. Merriman, and S. Osher. Motion of curves constrained on surfaces using a levet-set approach. *Journal of Computational Physics*, 175:604–644, 2002.



(a) (b) (c)

Figure 10. (a) Initial contours, and search spaces of width 2 surrounding them. (b) After one iteration of the MRC algorithm. (c) After five iterations of the regular MRC algorithm, and one subsequent iteration using refinement with $k = 1$.

[6] L. Cohen. On active contour models and balloons. *CVGIP: Image Understanding*, 53(2):211–218, 1991.

[7] D. Cohen-Steiner and J. M. Morvan. Restricted delaunay triangulations and normal cycle. In *Proc. 19-th Annual Symposium of Computational Geometry*, 2003.

[8] T. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1997.

[9] I. J. Cox, S. B. Rao, and Y. Zhong. "ratio regions": A technique for image segmentation. In *IEEE Int. Conf. on Pattern Recognition*, pages 557–564, 1996.

[10] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin. Modeling by example. *ACM Trans. on Graphics*, 23(3):652–663, 2004.

[11] D. D. Hoffman and W. A. Richards. Parts of recognition. *Cognition*, 18:65–96, 1984.

[12] I. Jermyn and H. Ishikawa. Globally optimal regions and boundaries as minimum ratio weight cycles. *IEEE Trans. on Pattern Analysis Machine Intelligence*, 23(10):1075–1088, 2001.

[13] M. R. Jung and H. K. Kim. Snaking across 3d meshes. In *Pacific Graphics*, 2004.

[14] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *Int. Journal on Computer Vision*, 1(4):321–331, 1987.

[15] M. Lanthier, A. Maheshwari, and J-R. Sack. Approximating weighted shortest paths on polyhedral surfaces. In *Proc. 13-th Annual Symposium of Computational Geometry*, 1997.

[16] Y. Lee and S. Lee. Geometric snakes for triangular meshes. *Computer Graphics Forum*, 21:229–238, 2002.

[17] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H.P. Seidel. Intelligent mesh scissoring using 3d snakes. In *Pacific Graphics*, pages 279–287, 2004.

[18] M. J. Milroy, C. Bradley, and G. W. Vickers. Segmentation of a wrap-around model using an active contour. *Computer Aided Design*, 29(4), 1997.

[19] K. Polthier and M. Schmies. Straightest geodesics on polyhedral surfaces. In *Mathematical Visualization*, pages 391–410, 1998.

[20] J. Shi and J. Malik. Normalized cuts and image segmentation. In *IEEE Int. Conf. on Computer Vision*, pages 731–737, 1997.

[21] S. Wang, T. Kubota, J. M. Siskind, and J. Wang. Salient closed boundary extraction with ratio contours. *IEEE Trans. on Pattern Analysis Machine Intelligence*, 27(4):546–561, 2005.

[22] S. Wang and J. M. Siskind. Image segmentation with minimum mean cut. In *IEEE Int. Conf. on Computer Vision*, pages 517–524, 2001.

[23] D. Williams and M. Shah. A fast algorithm for active contours and curvature estimation. *CVGIP: Image Understanding*, 55(1):14–26, 1992.

Mesh	Graph Resolution	Search Band Width	# Faces	Size of Cut	# Nodes	# Arcs
Horse	Regular	2	979	9	1356	3588
Horse	Refined: $k = 1$	2	979	34	8535	35833
Horse	Refined: $k = 2$	2	979	78	21332	125258
Hand (finger)	Regular	6	626	26	960	2629
Hand (thumb)	Regular	6	1456	26	2191	6061
Hand (thumb)	Regular	12	3039	50	4577	12999
Headless	Regular	12	3563	49	5386	15342
Venus (left eye)	Regular	2	644	9	964	2476
Venus (right eye)	Regular	2	740	9	1118	2853

Table 1. Statistics for the construction of search spaces and acyclic graphs. The number of faces is collected within the search space, on the original unrefined mesh. The number of nodes and arcs are collected on the constructed acyclic graph, which, given a fixed search space, would grow as the number of refinement levels k increases. These statistics are for the first iteration of optimization only.

Mesh	Max. # Neg. Iters.	Graph Construction	Min. Ratio	Time/Iter	# Iters.	Total
Horse: reg	4	0.10	0.01	0.11	1	0.11
Horse: $k = 1$	5	0.14	0.33	0.47	1	0.47
Horse: $k = 2$	6	0.22	2.47	2.69	1	2.69
Hand (finger)	5	0.00	0.01	0.01	2	0.02
Hand (thumb)	6	0.08	0.01	0.09	1	0.09
Hand (thumb)	6	0.17	0.15	0.32	1	0.32
Headless	6	0.21	0.18	0.39	2	0.78
Venus (left eye)	4	0.00	0.04	0.04	5	0.20
Venus (right eye)	5	0.00	0.06	0.06	5	0.30

Table 2. Timing statistics for our minimum ratio algorithm. Statistics for the construction of search spaces and acyclic graphs of the models are given in Table 1. Second column shows the maximum number of linear search steps needed per MRP problem. Third column shows time required for the construction of the acyclic graph, including finding the search space and edge cut. Fourth column shows time required to compute the MRC. Fifth column shows total time per iteration. Sixth column shows the total number of graph searches (iterations) performed to locate the final optimal contour. The last column shows the total time required. All timing statistics are averages if more than one iteration is required, and are measured in seconds.